Computer Science Department
CCIS, King Saud University

CSC 311 — Midterm I. — October 26, 2020.

The First Semester 2020/2021.

Duration: 90 Minutes — Points: 100

Q1:[25 points= 15+10]

(a) Using the definition of Θ, find $g(n)$ in the following : O, Ω, Θ
$2n^3 - 5n \in \Theta(g(n))$. Give a formal proof for your answer.

(b) Compare the order of growth of $f(n) = \log \sqrt{n}$ and $g(n) = \sqrt{\log(n^2)}$. Justify your answer.

Q2:[25 points= 12.5+12.5]

(a) Solve the following recurrence using the recursive (backward) substitution method and find $g(n)$, where $T(n) \in O(g(n))$. Justify your answer.

$T(n) = 4T(\frac{n}{4}) + 3n$

(b) Solve the following recurrence using the Master Theorem to find proper asymptotic bounds. Justify your answers.

$T(n) = 3T(\frac{n}{3}) + 3\log^3 n$ $\quad \log_3 3 = 1 >$ $\quad k = 0$ $\quad n^{\log_3 3}$

The Master Theorem:
Let $a \geq 1$ and $b > 1$ be constants, let $f(n)$ be an asymptotically positive function, and let $M(n)$ be defined on the nonnegative integers by the recurrence:
$M(n) = aM(\frac{n}{b}) + f(n)$,
where we interpret $\frac{n}{b}$ to mean either $\lfloor \frac{n}{b} \rfloor$ or $\lceil \frac{n}{b} \rceil$. Then $M(n)$ can be bounded asymptotically as follows.

1. If $f(n) \in O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $M(n) \in \Theta(n^{\log_b a})$. $>$
2. If $f(n) \in \Theta(n^{\log_b a} \log^k n)$, with $k \geq 0$, then $M(n) \in \Theta(n^{\log_b a} \log^{k+1} n)$. $=$
3. If $f(n) \in \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$ AND $af(\frac{n}{b}) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $M(n) \in \Theta(f(n))$. $<$

Q3:[25 points=10+15]
Consider the pseudo-code below:

n

```
int maximum_sum(int A[ ], int low, int high)
if high == low then              = 1
   return A[low];                = 1
end                                                              T   3.5
int mid = (low + high)/2 // Assume the floor function of the result;   1
int sum = 0;   = 1
for int i = mid; i >= low; i - - do       n/2                    | 3 | 1 | 2 | 6 | 5 | 4 |
   sum+ = A[i];
end
left_max = sum;   1
sum = 0 // reset sum to 0 ;
for int i = mid + 1; i <= high; i + + do     n/2
   sum+ = A[i];
end                                          2T(n/2) + n
right_max = sum;   1
int max_left_right = max(maximum_sum(A, low, mid), maximum_sum(A, mid + 1, high));
return max(max_left_right, left_max + right_max);
```

(a) What is the design technique used in this algorithm? Explain your answer.

(b) What is the time complexity of the following algorithm? Prove your answer.

Q4:[25 points]
An array of distinct integers $A[0..n-1]$ is called *convex* if and only if it satisfies the following condition: there exists an index c such that $A[0..c]$ is sorted in a non-decreasing order and $A[c..n-1]$ is sorted in a non-increasing order.
Example: {0,2,5,**10**,9,4,1}, {2,5,6,7,9,**13**}, and {100,80,50,**30**} are convex arrays.
Give the pseudo code of an algorithm that takes as input a convex array A and finds the largest integer in A in $O(\log n)$ time.

```
array convex = [0,1,2,......,n-1]
int max (A[ ], int first, last index {
 if (A[0] > A[1]) { return A[0] }
 else if (A[A.length-1] > A[A.length-2] ) { return A[A.length-1] ;
 else {
    int i = A.length/2 , i
    return max(A[ ], i , last index)
```